Test Automation Engineer- Technical Assessment

**Questions:**

**1) What other scenarios would you suggest for the given page? List 3-5 suggestions.**

A) Other suggested scenarios for the borrowing calculator page:

* Test with different income levels and validate the borrowing estimate.
* Test with different expense values (living expenses, loan repayments, etc.) and verify the impact on the borrowing estimate.
* Test with different values for total credit card limits and check if it affects the borrowing estimate.
* Test with different employment types (employed, self-employed, etc.) and verify the borrowing estimate based on the selected type.
* Test with different loan terms (e.g., 15 years, 20 years) and validate the borrowing estimate accordingly.

**2) If this test was part of a much larger test suite, how could you run a specific test or subset/group of tests?**

A) To run a specific test or subset/group of tests from a larger test suite, you can use test runners or tags provided by the testing framework. For example:

* In Cucumber, you can use tags to group related scenarios, and then specify those tags when running the tests through the command line.
* In TestNG, you can create test groups and include/exclude specific groups while executing the tests.

**3) Which approaches could be used to reduce the overall execution time? Describe how they could be implemented into your code base.**

A) Approaches to reduce the overall execution time:

* Implement parallel test execution: You can divide the tests into multiple threads or processes to run them concurrently, utilizing the available resources efficiently.
* Use test data optimization: Instead of generating test data on the fly, pre-generate and reuse the test data for multiple test cases to save time.
* Employ test data isolation: Ensure that each test case is independent and does not rely on the state of other test cases. This allows running tests in any order or in parallel without conflicts.
* Implement smart wait strategies: Use explicit waits to wait for specific conditions to be met instead of fixed sleep/wait times, thus reducing unnecessary waiting time.

**4) Sometimes UI tests can fail unpredictably. For example, the page may not have fully loaded before the automated test attempts to click a button on a web page. How would you improve reliability of these tests without increasing execution time?**

A) To improve the reliability of UI tests without increasing execution time:

1. Implement explicit waits: Use appropriate wait conditions to ensure that the page or specific elements are fully loaded and ready before performing actions on them.
2. Use unique element locators: Use robust and unique locators to identify elements on the page. This reduces the chances of false positives or failures due to incorrect element identification.
3. Handle asynchronous behavior: If the application under test has asynchronous operations, use appropriate techniques like waits, callbacks, or promises to handle and synchronize the interactions effectively.
4. Implement retry mechanisms: In case of intermittent failures, you can implement retry mechanisms to rerun the failed tests or actions to ensure stability.

**5) From your experience, what is the focus of UI automation testing – Integration, Functional or Acceptance testing? Briefly explain why.**

A) The focus of UI automation testing is primarily on Functional testing. UI tests verify that the application's user interface functions correctly and meets the specified requirements. They validate user interactions, data input/output, navigation, and the behavior of the application's UI components. UI tests are crucial in ensuring that the application works as intended and provides a satisfactory user experience. However, UI automation testing can also cover aspects of Integration testing when multiple components interact on the UI level and Acceptance testing when validating the application against user acceptance criteria or business requirements.